United States
Department of Agriculture

Office of the Chief Information Officer (OCIO)
Year 2000 Program Office

Year 2000 Testing Procedures Guide

Table of Contents

1.0 The Reason For Testing

The Year 2000 represents a threat to computer operating systems and applications, telecommunications, and embedded processors worldwide on a scale that has never before been experienced. As computer applications and embedded systems begin to encounter dates that occur after December 31, 1999, they will begin to misinterpret time-dependent events. Software experts estimate that about 5% of all U.S. businesses will go out of business because of this problem. In government, all sectors of the economy are at risk. Moreover, Year 2000 problems in one sector will have a domino effect on others due to the many interdependencies and linkages among them. In dealing with this complicated problem, many activities are underway to fix computer systems that are not Year 2000 compliant. One such activity in the Year 2000 life cycle is testing.

Testing Year 2000 modifications is crucial. Following are some of the many reasons why this testing is especially important:

- **The deadline cannot be delayed**. If a system isn't ready on 1/1/2000, no amount of negotiating or explaining will buy more time. A high percentage of software projects aren't completed on time – this can't be one of them.

- **There is only one chance to get it right**. On many projects, if the new version doesn't work, you can revert to the original system. Then you can correct the new version and reinstall it later. You can't do this when dealing with Year 2000 projects.

- **More applications and lines of code affected.** The Year 2000 project will likely affect more applications and more lines of code than any previous projects. The amount of testing needs to be correspondingly high.

- **Old code modified.** In many cases, code will be modifyed that hasn't been touched in years or decades. The original developers are probably long gone, and no one is familiar with the code. This alone dictate that testing is especially thorough.

- **Systems in every area of life affected.** The Year 2000 problem affects systems as well as the computers of every organization you deal with. It is important to test all data coming into the system as well as data getting sent out.

Since the Year 2000 computing problem is so pervasive, potentially affecting Agencies' systems software, applications software, databases, hardware, firmware, telecommunications, external interfaces, and vulnerable systems and processes, the requisite testing is extensive and expensive. Experience is showing that Year 2000 testing is consuming between 50 and 70 percent of a project's time and resources. Therefore, complete and thorough Year 2000 testing is essential to provide reasonable assurance that new or modified systems process dates correctly and will not jeopardize an Agency's ability to perform core business operations after the millennium. Until thorough testing has been performed, there is no certainty that changes made are complete or correct. No one wants any nasty surprises on the morning of Monday, January 3, 2000.

A problem reporting system should be developed to track the status of Year 2000 problems identified during testing.

2.0 Estimating Testing Time

The amount of time devoted to testing will be a significant portion of your Year 2000 effort. Year 2000 consultants and vendors estimate that testing will represent from 50 - 70 percent of the total time spent on these projects. If your project plan doesn't allocate roughly this amount of time for testing, you should reexamine it.

The Year 2000 plan for the state of California for instance, estimates that testing will be 40 percent of the cost of the project. Table 1 is an illustration of cost and time estimates by project phase for the state of California.

Table 1. *California's Year 2000 Project Estimates*[1]

Project Phase	Percent of Project Cost	Time Frame (in months)
Awareness	1	1
Inventory	1	1
Assessment	5	2
Solution design and handling	15	3-4
Development and modification	20	6-8
Testing	40	6-8
Implementation	10	4-5
Monitoring	8	6-10

3.0 Test Early and Test Often

"Vote early and vote often" was the advice on a campaign banner in the 1850s. This advice should be taken to heart, although in a slightly different context, during any Year 2000 project. Following are two of the many positives to testing early:

- Testing will almost certainly take longer than expected. Starting early gives a little more assurance that the project will finish in time.

- Testing early exposes problems earlier. Use this extra time to correct problems. If early testing exposes shortcomings in a project plan, procedures, or training, the earlier they are discovered the more time available to correct them.

Testing also has many advantages. This project will affect a number of software components, such as operating systems, compilers, assemblers, link-editors, applications, and JCL scripts. Not all will be modified at the same time. Any one of these pieces has the potential to affect earlier changes.

For example, suppose that modifications have been made to a COBOL program and it passes the acceptance test. What happens when a new Year 2000 compliant version of the COBOL compiler is installed on the mainframe and it is used to recompile all of the programs? How can we be certain that it won't introduce errors into a program that was working previously? The only way to be certain this doesn't occur is to retest each program. This form of testing is called regression testing.

4.0 Planning Your Testing Activities

As part of the Agency's establishment of an enterprise Year 2000 program strategy, a Test and Evaluation Master Plan (TEMP) should be developed, issued, and continuously updated to reflect changes in the test Agency's structure, actual test progress and experience, actual resource availability, and changing priorities.

Earlier steps in any Year 2000 project will have broken the effort down to manageable-sized chunks. These chunks are sometimes called upgrade units or partitions. Each Agency needs to create a formal TEMP for each piece. This section lists the components that should make up your TEMP.

4.1 Acceptance Tests

The objective of testing is to make certain that software performs as expected. The acceptance test plan needs to lay out, in great detail, the steps of each test. These steps need to be clear enough that anyone can perform them correctly every time.

It has been said that if you are a trial lawyer: *Never ask a witness a question if you don't know how it will be answered.* The wisdom of this advice also applies to software testing. Successful testing means knowing in advance what the results of a test will be. A test plan must describe the tests that will be run as well as the expected outcome. Otherwise, how will we know if the test has been passed?

4.2 Responsible Parties

When establishing your program management structure, a Year 2000 test manager (program level) should be designated and given the authority and responsibility for ensuring that Year 2000 testing is planned, conducted, and reported in a structured and disciplined fashion, and is independently reviewed by an independent verification and validation agent. A number of members of the Year 2000 team will probably be involved in the testing phase.

Generally testing falls into three categories, **minimal testing**, which covers only the basic functionality and Year 2000 date impacts, **nominal testing** for the applications and embedded system processes that would ensure full functionality and Year 2000 compliance for primary date calculations and **comprehensive testing** which considers things like upper and lower boundary checking, erroneous data checking all possible Year 2000 impacts and all application functionality.

Minimal testing must be performed on all mission critical systems to ensure Year 2000 compliancy. However, all test plans should consider whenever practical comprehensive testing for all possible Year 2000 impacts and all application functionality. A phased approach may be appropriate where on the first pass only minimal testing takes place. When all applications have completed the minimal testing then on the second pass the comprehensive tests are conducted. This approach may be necessary when time becomes an issue and all systems have not been tested.

A testing manager should be appointed and at a minimum (1) define and assign roles, responsibilities and expectations for Year 2000 testing, (2) define criteria for certifying a system as Year 2000 compliant, (3) develop

and maintain a test and evaluation master plan, (4) establish independent quality assurance/verification and validation of test activities, (5) obtain necessary test budgets, (6) establish test environments/facilities, augment them as needed, and schedule their use according to established priorities, (7) follow the test guidance provided and where necessary provide further detailed guidance for the various types of testing (software unit/module, integration, system acceptance, and end-to-end), and ensure that this guidance is followed, (8) establish support processes and resource/information sources, (9) establish a formal test activity and progress reporting requirements, (10) and establish a library of test tools. The manager will assign the roles and responsibility of each team member on each test. Some roles follow:

- Designing the test. This is a very time consuming and often tedious job that requires a good knowledge of the application, the expected results and the conditions under which the test must pass.

- Preparing the test data. Test data must be developed to ensure that all conditions are tested. Real data should never be used in testing because all operating conditions may not be revealed with relatively clean data. Data is developed that is error plagued to see how the program reacts. Are error messages invoked, how does the program react, is the result what was expected?

- Executing the test. The tests must be run over and over again to test all conditions. Of special concern is the conditions under which the tests are run. NEVER run tests in the production environment unless strict and absolute controls are in place to ensure the day-to-day production system is not affected and the test data doesn't contaminate real production data.

- Verifying that the test was successful. Verification is essential to determine that all tests run successfully and that errors were handled as required. Caution must be exercised to document the conditions and ALL errors handled. Whenever possible automated test tools, that allow the user to modify the test cases and parameters of the tests, should be used. They not only save time but they are easily repeated when tests have to be re-run for regression testing.

- Signing off that the test was successful. This step requires the person performing the test to sign off and certify the completeness of the tests. It is recommended that all testing be certified by a reviewer and not the actual tester. This ensures that all tests were run and that the results were documented.

- Logging the results. ALL testing activities must be recorded. This includes the scripts, logs, error messages, anomalies reported and the data files, and anything else that has significance and would be necessary to duplicate the test in the future.

- Assuming control of the source code after it passes the test. The modified and now Year 2000 compliant source code must be archived. The new source code must also be retained for future testing as other changes are made in the future.

4.3 Test Preparation

Effective testing requires that a number of pieces come together successfully. Someone has to make sure that these pieces are all in the right place at the right time. Some of the pieces follow:

Reserve test hardware for given tests. It may not be possible (or advisable) to run multiple tests on the same hardware simultaneously. Schedule time on the hardware just like you schedule conference rooms.

Properly configure test hardware, making sure that the appropriate network connections, disk devices, modems, multiple terminals (to simulate multiple users), tape drivers, and so on exist.

Load all required software onto the test system, including the operating system, test tools, remote access software, and date simulation program.

Make sure the most recent version of the software to be tested is available. This task must be carefully checked if you have many testers. Previous test efforts might have left uncertified software on the machine. Executing the wrong version of software wastes testing times and is extremely frustrating.

Make sure that the correct test data is in place. Test data might exist for multiple time periods. Be certain that the desired data is loaded onto the system when tests are performed.

Check the availability of all needed consumable items, such as computer paper, tape cartridges, and floppy diskettes.

Have any required logging or tracking software in place. It's a waste of time to have to repeat tests because you didn't capture logging information.

If local etiquette requires it, remove any software or data you loaded after you finish. This reduces the chance that subsequent tests will inadvertently run with your software or data.

5.0 Testing Year 2000 Changes

Testing software changes involves several different phases. This is true for development projects, Year 2000 changes, and general fixes and enhancements. Each testing phase is geared toward testing certain types of errors. The combination of all types of testing is more likely to result in reliable software.

5.1 Additional Hardware for Testing

You might have to buy or lease additional hardware for your Year 2000 testing. One example is if you can't risk changing software on a production system. Crashing a production job while running a test program isn't going to win you any friends with management, production staff, or users.

Another potential effect on production applications is performance. If systems are operating at their capacity now, adding testing activity might be the straw that breaks the camel's back. Extra processing might increase the response time that users experience to unacceptable levels. Batch jobs might not complete in their available windows.

Your current disk space might not be sufficient to add extra programs and data. Test data can be a significant consumer of storage space. To add new versions of programs and test data, you might have to add more disk drivers.

Your business environment might require that Year 2000 activities be performed during nonbusiness hours. This will require the team to work nights. Besides disrupting people's lives for a year or two, a schedule shift has other negative effects. While working nights, they won't be able to easily communicate with other MIS staff or users. This will cause problems if emergencies arise during the day while they are home asleep. If hardware is available that is dedicated to testing, your team can test during normal business hours.

5.2 Involving Users in the Testing Process

Users need to be involved in the Year 2000 effort. Involving them in this process has definite advantages:

- Project becomes "our" project.
- Users can provide insight into where dates are used.
- Users know how the system is supposed to work.
- In testing the system, users can make significant contributions.

The most important reason for involving the users in the testing process is to add validity to the testing effort and to have access to the knowledge the uses has on the system performance parameters.

It will be extremely difficult to build a valid set of tests, test scripts and the data that will insure ALL conditions are tested.

Testing software changes involves several different phases. Each testing phase is geared toward testing certain error types. The combination of all types of testing is more likely to result in reliable software.

5.2.1 Unit Testing.

Unit testing is the act of testing a single program. Unit tests check the correctness of Year 2000 changes at the program or module level. Any module modified during the remediation effort must be unit tested. The programmer who modified the module typically executes these tests. Tests are conducted using debuggers, execution simulators, and interactive testing tools. The programmer creates small test data files for debuggers or selects decision points in an execution simulator. Validation is performed manually by the programmer.

For specific test scenarios on unit testing, refer to the GAO Year 2000 Computing Crisis: A Testing Guide.

5.2.2 Integration Testing.

Integration testing checks the interfaces between sets of modules. Performed by the test team with coordination from the Quality Assurance (QA) group to all system owners, especially when the integration tests are being run on external interfacing systems. The testing is best accomplished by using a variety of data and observing how the application

responds. Checking interfaces is especially important for century-compliance efforts as dates are typically passed across module interfaces. During integration testing, individual modules are combined into increasingly larger groupings (by function, related functions, subsystem, etc.). The test data used for integration tests concentrates on the issues that are caused by mismatched versions or incorrect module interfaces rather than attempting to achieve complete test coverage. Integration tests can use the tools and techniques for unit testing for small integration groupings and system testing techniques for larger groupings.

For specific test scenarios on integration testing, refer to the GAO Year 2000 Computing Crisis: A Testing Guide.

5.2.3 System Acceptance Testing.

Similar to integration testing, but at a higher level. It tests the entire system at one time instead of a single part of the system. These tests are typically performed by the testing organization. System tests cover all facets of application functionality and are designed to mirror the application's production operation. These tests will be conducted using past, current, and future dates. They are validated by comparing the results of parallel runs of the compliant and noncompliant versions of an application.

For specific test scenarios on system acceptance testing, refer to the GAO Year 2000 Computing Crisis: A Testing Guide.

5.2.4 Regression Testing.

Regression testing exposes problems that occur from changes made to other modules of the system. Each regression test requires that every feature in an application be tested again. Regression tests are designed to ensure that existing functionality has not been affected by the Year 2000 migration. In theory, achieving century-date compliance should have no effect on application functionality other than adding the ability to handle century dates. Regression tests are run using the current date, and the results of the tests are validated through comparisons of parallel runs of compliant and non-compliant versions of the application.

For specific test scenarios on regression testing, refer to the GAO Year 2000 Computing Crisis: A Testing Guide.

5.2.5 End-to-End Testing

The purpose of end-to-end testing is to verify that a defined set of interrelated systems, which collectively support an organizational core business area or function, inter-operate as intended in an operational (i.e., live production) environment. These interrelated systems include not only those owned and managed by the Agency, but also the external systems with which they interface. For example, since Agencies that administer key federal benefits payment programs, such as the National Finance Center, exchanges data with the Department of the Treasury which, in turn, interfaces with various financial institutions to ensure that all government payroll checks are issued, end-to-end testing of the federal payroll function would include systems for all entities involved, as well as their supporting telecommunications infrastructures.

For specific test scenarios on end-to-end testing, refer to the GAO Year 2000 Computing Crisis: A Testing Guide.

6.0 Creating Comprehensive Test Data

Unless complete regression test libraries are available, test data must be created to support Year 2000 testing. This function encompasses all of the activities and tools required in creating, manipulating, and managing test data. Although Year 2000 projects have complex test data requirements, careful planning and reuse during test data creation can reduce the effort required to obtain adequate test coverage. There are three major methods for creating test data: *extracting production data, using a test data generator, and manually creating test data*. Extracting production data is typically the easiest approach, but often results in low levels of test coverage. Test data generators and capture/replay tools create test data based on input parameters or by capturing keystrokes. Initial set-up can be laborious for use in building complete regression libraries, but is useful for building specialized test cases. Once created, test data can be reused for multiple types of tests. For example, regression test data can be forward dated for use in future date testing. Each year field in the regression test data can be incremented by 28 to move it into the future (e.g., 1996 becomes 2024). The use of 28 guarantees that all dates fall on the same day of the week.

6.1 Data Quality

The quality of the test data can make or break the testing activity. All testing data should be closely examined to insure that it would provide complete valid testing. The test data must be of sufficient quantity and

diverse to provide the necessary tests that could be encountered in an actual production environment. Good test data must be similar to production data both in quantity and diversity

6.1.1 Quantity

There should be sufficient quantity of test data files to simulate the proper loading of the tested system. All to often testing activities no not properly load the system with enough test data to simulate real loading characteristics and that must be avoided.

6.1.2 Diversity

Data that isn't diverse can make a program look like it's correct, when it isn't. Some of the data must be unusual or outright incorrect to fully test the system / program. Diversity of languages, platforms, operating system, databases, and other application environment tools and utilities presents the greatest technical challenge for enterprise-level testing. Each of these environments must be tested and certified before the applications that operate within them can be certified. There are no standard testing tools that can be used across all environments. Most tools are limited to one or two languages on a single platform. Further, the number and quality of the application testing tools available on the market decrease quickly as the platforms become less common.

6.2 Specific Year 2000 Data

Test data for a Year 2000 project must reflect that goal. In order to insure the programs are executing properly numerous variations of month, day and year formats are required in our testing data files. Some examples are:

12/31/1998
01/01/1999
09/09/1999
10/01/1999
12/31/1999
01/01/2000
½/2000
01/03/2000
01/10/2000
02/28/2000
02/29/2000
03/01/2000
10/10/2000

12/31/2000
01/01/2001
02/28/2001
02/29/2001 invalid date
03/01/2001
12/31/2001
01/01/2002
02/29/2004

6.3 Generating Test Data

- Generate the test data manually or
- Obtain a data-generating tool or
- Capture existing production data and manually modify the dates to test the year 2000 situation.

6.3.1 Day-of-week Calculations.

Check the day-of-week (DOW) calculation. This is used for tape backups, report generation and when files or tapes are deleted.

6.3.2 Windowing Data

Windowing is a popular remediation repair strategy and one that is used widely. Always determine the size of the window and then test both within the window and outside the window. As an example: if the system uses a 100-year window, test with the following dates to assure correct display areas:

December 31, 1929, January 1, 1930
December 31, 2029, January 1, 2030

6.4 Examining Other Aspects of Testing

6.4.1 Leap Year.

2000 is a leap year and February 29th is a good date. This must be tested. Check the 1st of March to be a Wednesday with DOW software.

6.4.2 Third Party Software.

Third party software should be tested thoroughly and test data and test cases should be created. Any errors found need to be identified and given to the vendor for correction immediately.

6.4.3 Backup Procedures

Backup and restoration procedures are very dependent on dates and date processing. Restoration processes are also heavily date dependent. The backup must restore the most recent complete backup. It then needs to apply the incremental backups for every day since the total backup. Dates and day of week (DOW) logic drive all this information. Test the backup and restoration procedures.

6.4.4 Integrating Other Changes

After changes have been made and testing is completed, regression testing needs to be performed to assure there are no errors.

7.0 Testing Tools.

Testing without test tools is not practical. The amount of work to do is simply too great to perform manually.

7.1 Debugger.

The debugger is a test tool that enables a programmer to observe exactly what happens as a program executes by inserting breakpoints in the source code. When a breakpoint occurs, the programmer can “look around” the program and do the following:

- Examine contents of variables
- Alter the contents of variables
- Modify the source code from within the debugger
- Display the contents of any memory location.

7.2 Date Simulator.

Date simulators simulates the system time on a computer, usually a mainframe computer. Date simulators operate by intercepting system date requests from applications. Usually, date simulators maintain a list of programs that should be given the simulated time. All other calling

programs are passed the true system time. This makes it possible to test a specific set of programs at a given time.

- Changing the system time on a mainframe frequently requires an IPL (Initial Program Load).
- If the system time is set forward, accounts and their passwords may expire. This problem may cost more time than it's worth.
- Product licenses of third-party software may expire when the system time is set forward. Simply resetting the time may violate the license agreement. Since 1972 is exactly the same as the year 2000 in terms as the number of days and the day of the week each day fallls on. Then setting the date backward to test the leap ear feature of 2000 will preclude loosing licenses
- Frequently, files and data sets are given an expiration date when they are created. These files may be automatically deleted during testing. Change the expiration date to a time in the future, i.e. 1/1/2020.

7.3 Data Aging Tools

Data aging tools age the data. To use, copy production data to a separate file, volume. The data is aged, say 20 years and tests are run on the data.

7.4 Capture and Playback Tools

Capture and playback tools allow reproducing test sessions.

- Test sessions are replayed exactly
- Replays of test sessions run much faster
- Test sessions can be run unattended
- If results are different, the differences are logged.

7.5 Coverage Analysis Tools

Estimates of the percentage of code that will be directly affected by the year 2000 problem range from 1 to 5%. A medium sized agency may have 10,000,000 lines of source code. The lines affected range from 100,000 to 500,000. Even if the lower estimate is correct, that's a lot of source code.

- Produces such as, Workbench/2000 and VIA/Smarttest provide coverage analysis. The results can be summarized and printed.

7.6 Performance Analysis Tools

Year 2000 compliant programs will execute slower than the original versions for a number of reasons:

- Additional logic has been added to programs
- Data records may be longer
- Data will be read in smaller blocks

Performance analysis is important only when the changes made as a Year 2000 remediation effort degrade the performance to an unacceptable level. This problem can be solved. However, it is usually expensive because it often requires a hardware upgrade.

APPENDIX A

References

The following list of references is pertinent to the Year 2000 project.

1. ANSI X3.30 - Formatting Date Data

2. FIPS-4-1 (Revised 1996-03-25) - Federal or DOD procurements

3. ASC X12 EDI draft STD for trial use, ISO 9735, UN/EDIFACT - Electronic commerce (EDI)

4. The Modified Julian Date (MJD) has been officially recognized by the International Astronomical Union (IAU), and by the Consultative Committee for Radio (CCIR), the advisory committee to the International Telecommunications Union (ITU). The pertinent document is CCIR recommendation 457-1 use of the modified Julian dates by the standard frequency and time-signal services. This document is contained in the CCIR "Green Book" Volume VII.

5. The Almanac for Computers also provides information on JD & MJD.

6. Additional, extensive documentation regarding the Julian Date (JD) is contained in the Explanatory Supplement to the Astronomical Ephemeris and Nautical Almanac, and in the yearbooks themselves, now called The Astronomical Almanac.

7. Testing Very Big Systems; David Marks, McGraw Hill 1992, ISBN 0-07-040433-X

8. Software Testing; Marc Roper, 1994, ISBN 0-07-707466-1

9. Testing Computer Software; Kaner, Tab, 1988, ISBN 0-8306-9563-X

10. Software testing in the Real World, improving the process; Edward Kit, Addison-Wesley, 1995, ISBN 0-201-87756-2

11. Software Testing A Craftsman's Appr; Paul C Jorgensen, ISBN 0-8493- 7345-X

12. IEEE Standard for Software Verification and Validation Plans (IEEE Standard 1012

13. IEEE Standard 829-1983 for Software Test Documentation (updated 1991)

14. Year 200 Solutions for Dummies, K.C. Bourne, IDG Books Worldwide, Inc., 1997.

APPENDIX B

Glossary

The following list of words may have special meaning in the context of the Year 2000 project.

Calendar errors — Errors typically include failing to treat 2000 as a leap year and converting incorrectly between date representations.

Cluster — A cluster is composed of multiple systems that constitute a complete process.

Combined Component — Any stand-alone, computer-based, commercial off-the-shelf device or software package that has two or more date/time functions that can affect test results.

Compliance — Year 2000 compliance means that neither performance nor functionality is affected by dates prior to, during and after Year 2000. Compliance will be demonstrated when the criteria of General, date, and century integrity are satisfied.

Component — Any stand-alone, computer-based, commercial off-the-shelf device or software package that has only one date/time function that can affect test results. The smallest unit of testing for the Year 2000 project.

Cross Cluster Testing — The highest level of integration testing which is organized around a particular function or across functions. An example is a product order through product delivery.

Date integrity — All manipulations of calendar-related data (dates, duration, days of week) will produce desired results for all valid date values within the application.

Date overflow — Many software products represent dates internally as a base date/time plus an offset in days, seconds, or microseconds since that base date/time.

	Hardware integers holding the offset value can overflow past the maximum corresponding date—an event, which may lead to, undefined behaviors.
Explicit first 2 digits of year	Date elements in interfaces and data storage permit specifying the first 2 digits of the year to eliminate date ambiguity.
Extended semantics	In general, specific values for a date field is reserved for special interpretation. The most common example is interpreting "99" in a 2-digit year field as an indefinite end date, i.e., "does not expire." Another is embedding a date value in a non-date data element.
First 2 digits of year ambiguity	This is the most common element. Software represents dates with a 1- or 2-digit year. When software does not recognize that dates are not all in the 19xx range, the results are undesirable.
General integrity	No value for current date will cause interruptions in normal operation.
Gregorian Calendar	Revision of the Julian calendar in 1582 by Pope Gregory XIII, adopted by the US and Great Britain in 1752. Added that centesimal leap years must be divisible by 400 rule and suppressed 10 or 11 days during 1700.
Inconsistent semantics	At interface between systems, software on each side assumes semantics of data passed. Software must make same first 2 digits of date assumptions about 2-digit years.
Independent Verification & Validation	The process of an entity verifying and validating findings of another entity while staying completely neutral as to the outcome of the process.
Julian Calendar	Introduced in Rome in 46 B.C., it established a 12 month year of 365 days with every 4th year having 366 days.
Julian Date	Julian Date (JD) is the number of days since Noon 4713 BC plus the fractional part of a day for the

	time of day past Noon.
Modified Julian Date	Number of days to 5 digits and shifted the beginning of the time of day to Midnight. For 1997, the MJD is 50448 + DOY (day of year).
Modified Julian Date	(MJD) is the Julian Date minus 2,400,0005 which reduced the size of the Real-time Clock A battery operated clock which keeps time when the system is powered off Virtual Clock. A software based clock used in some operating systems to maintain the time and date as an operating system service.
System	A system is composed of multiple combined components and/or components that form part of a operating process, i.e. pay roll program, Production Cell.
Test plan	A test plan is a documented set of test cases and test scripts.
Test case	A test case is a documented test procedure with specific input data and expected test results. Example - rollover Dec 31, 1999
Test procedure	A test procedure is a step by step description of the test to be performed. Example – rollover
Unit	A Unit is the minimum recognizable level to which equipment containing a date/time function or processor can be broken down. The Inventory will be composed of multiple Units.
Year 2000 Ready	There will be no impact on production nor product quality due to Year 2000 date issues, but compliance is NOT required.

APPENDIX C

Various Test Procedures

Several test scenarios have been developed as a result of problems identified with the year 2000. This limited set of tests cannot prove a component/system to be Year 2000 compliant, but using them will help identify several frequently observed problems. These test procedures are written as general instructions. Specific knowledge of the systems or components under test is also required in order to apply these test cases.

A brief description of each test is provided for guidance in conducting tests on systems with unknown status.

The following test procedures provide step by step instructions for performing each test. The results should be recorded step by step as the test is performed to ensure accurate records of the test are documented on the "Year 2000 Test Report" form. The test results should be retained locally.

1. Critical Date Values for Year 2000 Testing

The following dates should be tested for proper operation:

1.0000-00-00 Special Value
2.1998-12-31 Rollover, Reboot
3.1999-01-01 Special Value
4.1999-09-09 Special Value
5.1999-12-31 Special Value, Rollover, Reboot
6.2000-01-01 Day of Week, Day of Year
7.2000-02-28 Rollover, Reboot
8.2000-02-29 Rollover, Reboot, Day of Week
9.2000-03-01 Day of Week
10.2000-12-31 Rollover, Reboot, Day of Week, Day of Year
11.2001-01-01 Day of Week, Day of Year
12.2027-12-31 Rollover, Reboot, Day of Week, Day of Year

2. Rollover, Reboot, Day of Week Tests

The rollover test checks for proper handling of the date transition from 1999 to 2000 without manual intervention. Based on actual tests several different results have been observed as examples of incorrect handling of the transition from 1999 to 2000. Many systems used 2-digit dates and the result may be a rollover to year 100; sometimes the 19 is assumed and the result is the year 19100. For other unknown reasons the years 2001, 2028,

and non-printable characters have been observed. The effect of incorrect date calculations may include negative numbers.

The reboot test checks for correct date & time storage during power cycles of the system. The system may function correctly when the time is set ahead, but revert to another time and date when the power is cycled. Many PC's revert to 1980 or 1984 when rebooted after the year 2000.

The day of the week may be incorrectly calculated. Systems should display the day of the week of January 1, 2000 as Saturday, not Monday, which may mean January 1, 1900.

A. Rollover - 1999 to 2000 - Power on

Test:

- Set the date to 31 Dec. 1999.
- Set the time to 23:59 (11:59 p.m.).
- Observe the system date after 00:00 am

Expected Result:

- The system clock advances into the year 2000 and continues normally.

B. Day of Week

Test:

- The clock is set to 1 Jan 2000.
- Observe the system day of week display.

Expected Result:

- The system displays the day of week as Saturday. (1 Jan 1900 was a Monday)

C. Reboot - Date retention

Test:

- Set the date to 1 Jan 2000.
- Power down the system.
- Power up the system.
- Observe the system date

Expected Result:

- The system clock still displays the year 2000 and operates normally.

Note: Many personal computers reset themselves to 04 January 1980, or some other past date, whenever they reboot, if the CMOS real time clock says the year is 00.

D. Rollover - 1999 to 2000 - Power Off

The procedure specifies 10 minutes before mid-night, but a smaller time may be appropriate. Be sure you can shutdown the system before the rollover occurs.

Test:

- Set the date to 31 Dec. 1999.
- Set the time to 23:50 (11:50 p.m.).
- Power down the system before it can roll over to year 2000
- Wait until after midnight with the power off.
- Power up the system.
- Observe the system date

Expected Result:

- The system clock advances into the year 2000 and operates normally.

3. Manual Date Set Test

These test checks for correct date & time entry to initialize the system clock. The "set system date" function may operate incorrectly when the time is set ahead, not allow entry over a certain date range, or revert to another time and date when set. Some PC's revert to a default date (1980 or 1984) when set to a date in the year 2000. Some systems have multiple date set functions; for a PC the date may be set using the CMOS Setup program at power on, using a DOS date function, or using a windows clock or control panel interface. The tests in this section should be executed on all date set functions for the system.

The date set function may also be accessed through an application-programming interface (API). If the equipment has a battery backed up clock, the date set test may include removing both battery power and external power to completely initialize the system clock and attempting to set the date to 1 Jan 2000. Exercise caution to document all system configurations when attempting this test because the configuration may be lost upon removal of the battery.

A. Date Set - 1 Jan 2000

Test:

- Set the date to 1 Jan 2000.
- Observe the system date

Expected Result:

- The date should be Saturday, 1 Jan 2000.

B. Date Set - Date retention

Check to insure that the date set function sets the real-time clock, not just the system's virtual clock.

Test:

- With the date still in the year 2000, power down the system.
- Power up the system.
- Observe the system date

Expected Result:

- The system clock still displays the year 2000 and operates normally.

Note: Some PC's which fail the Reboot - Date retention test will pass the manual Date retention test. This is a possible fix for those PC's.

C. Date Set - 29 Feb. 2000

Test:

- Set the date to 29 Feb. 2000.
- Observe the system date.

Expected Result:

- The date should be Tuesday 29 Feb. 2000.

D. Leap Year Test

The leap year test checks the logic, which calculates valid dates for leap year. An example of a failure on leap year was published on the Internet, which told of 66 industrial controllers in a steel mill all locking up when the date calculation for leap year 1996 occurred. A 2-digit year representation presents a possible divide by zero problems. The year 2000 leap year calculation is more

complex because multiple exceptions apply to the calculation, leading to greater opportunities for error. The following are leap year considerations:

- if the year is divisible by four, it is a leap year;
- if the year is divisible by 400, then it is a leap year;
- if the year is 3600, it is not a leap year.

E. Leap Year - Rollover 2/28 - Power On

Test:

- Set the date to Monday 28 Feb. 2000.
- Set the time to 23:59 (11:59 p.m.).
- Observe the system date after midnight

Expected Result:

- The date should be Tuesday 29 Feb. 2000.

F. Leap Year - Reboot 2/29

Test:

- Set the date to 29 Feb. 2000.
- Power down the system.
- Power up the system.
- Observe the system date

Expected Result:

- The date should be Tuesday 29 Feb. 2000.

G. Leap Year - Rollover 2/29 - Power On

Test:

- Set date to 29 Feb 2000
- Set the time to 23:59 (11:59 p.m.).
- Observe the system date after 00:00 am

Expected Result:

- The date should be Wednesday 1 March 2000.

4. Date Window Tests

Windowing date systems assume the first 2 digits of a 4-digit year to be 20 for values below the switch value and 19 for values above the switch value. An example switch value of 50 provides for a range of 1951 to

2049. If the 2-digit year is greater than 50 the year is assumed to be 19xx. That is, 84 is greater than the switch value so the year is 1984. If the 2-digit year is less than 50 the year is assumed to be 20xx. That is, 34 is less than the switch value, so the year is 2034. When two integrated systems share date information in this format be sure to test the interface at the boundary conditions. Is the behavior specified when the year is the switch value? Do both sides of an interface switch the same way?

Systems using date windowing should consider testing:

Creation of date data at the switch boundary dates, above and below; Modification of configurable windowing parameters, i.e. changes the switch value; Modified switch boundary dates, above and below.

A. Date Window Test - Below Limit

Test:

- Observe the configured switch value.
- Change the current date to one year below the switch value.
- Observe a 4-digit date.

Expected Result:

- The date assumes 20xx.

B. Date Window Test - Above Limit

Test:

- Observe the configured switch value.
- Change the current date to one year above the switch value.
- Observe a 4-digit date

Expected Result:

- The date assumes 19xx.

C. Date Window Test - Change Limit

Test:

- Change the configurable switch value to 2004.
- Observe the configured switch value.

Expected Result:

- Limit has been changed to 2004

(Repeat the above and below limit tests to confirm the limit has changed.)

5. Other Date Representation Tests

The following date formats occur often enough that a brief description is included to stimulate thinking about possible date related functions and interface definitions:

Julian Date (JD) - a real number where the integer part represents the number of days since 12:00 Noon 1 January -4712 (Julian day zero) and the fraction part is the part of the 24 hour day past Noon;

Modified Julian Date (MJD) - the Julian Date minus 2,400,000.5 shifting the reference date to Midnight 17 November 1858. This reduces the number to 5 digits for the next 150 years. In 1997 the MJD = 50448 + DOY; Gregorian Date in Day of Year formats (DOY); Dates represented using a YYDOY or YYYYDOY format where Y is the Year in 2-digit or 4-digit format and DOY is the day of the year from 001 to 365 (or 366 on a leap year) should be tested for correct operation.

The following DOY dates should be checked:

- 29 February 2000 should be 00060 or 2000060;
- 31 December 2000 should be 00366 or 2000366;
- Invalid Dates like 98000, 98367, 00000, and 2000000;
- Special Codes 99365.

A. DOY - 29 February 2000

Test:

- Set the date to 29 February 2000.
- Observe the DOY date by function call or system display.

Expected Result:

- 29 February 2000 should be 00060 or 2000060.

B. DOY - 31 December 2000

Test:

- Set the date to 31 December 2000.
- Observe the date by function call or system display.

Expected Result:

- 31 December 2000 should be 00366 or 2000366.

C. DOY - Invalid Dates

Test:

- Attempt to set the DOY date to 2000000.
- Observe the DOY date function call return value.
- Attempt to set the DOY date to 98367.
- Observe the DOY date function call return value.

Expected Result:

- Error codes or messages as documented by vendor.

6. Arithmetic Date Tests

If dates are used in any calculations, test for correct operation. The following list is intended to help identify functions, which should be checked:

- Period calculations;
- Financial functions based on a time period;
- Shelf life calculations; Time Remaining.

A. Days in 2000

Test:

- Create a period calculation using 1-Jan-2000 as the start date and 31-Dec-2000 as the end date.

Expected Result:

- The year 2000 has 366 days.

B. Days across 1999/2000 Boundary

Test:

- Create a period calculation using 1-Dec-1999 as the start date and 31-Jan-2000 as the end date.

Expected Result:

- The period ((31 January 2000)- (1 December 1999)) has 61 days.

C. 3 Days across leap year

Test:

- Create a period calculation using 1-Feb-2000 as the begin date and 1-Mar-2000 as the end date.

Expected Result:

- The month of February has 29 days.

7. Upload / Download Tests

The upload and download tests check for logic, which prevents new files from replacing old files when the date comparison uses only 2 digits. The logic must be reversed when the dates cross the year 2000 boundary. Test to see if a file created in (19)99 is considered older than a file created in (20)00. Systems have failed to down load new programs because they were assumed to be older that the current program on the system.

Old File Date	New File Date	2-digit difference	4-digit difference
July 4, 1998	July 4, 1999	+1	+1
July 4, 1999	July 4, 2000	-99	+1
July 4, 2000	July 4, 2001	+1	+1

A. Upload

Test:

- Set the date of the control system to January 11, 2000.
- Attempt to upload the test file.

Expected Result:

- Verify that the new file was uploaded.

B. Download

Preparation:

- Download the existing test file with a date prior to January 1, 2000.

- Create a new version of the test file to download with the file date January 5, 2000.

Test:

- Set the date of the control system under test to January 10, 2000.
- Attempt to download the January 5, 2000 test file.

Expected Result:

- Verify that the new file was downloaded.

8. Special Value Test

The special value test checks for usage of values in date fields for special purposes that are not dates. An example is special handling of the date September 9, 1999, which may be used as a special code for software license expiration dates, or never expires codes, and/or errors. Systems integrated to higher level systems should be subjected to special value tests. Special values considered should include the date values 9-9-99, 0-0-00, and x-x-9999. This test applies to applications, which create records containing the current date as a time or data field, such as database applications, or systems, which maintain historical data.

Test:

- Set the current date to a special value (e.g. 9-9-99, 9-9-1999, 0-0-00, and 0-0-0000).
- Observe the number of records in a test file at the start of the test.
- Using the application under test, create a new record that contains the current date.

Expected Result:

- Observe that the application was able to create the test record.
- Observe that the test record is included in displays or reports as applicable.
- Observe that the end of file continues to function correctly (e.g. number of records correct?).
- Observe that the test record can be deleted from the system.

Examples of failure include:

- Not terminating an expired software license;
- Failing to age backup tapes for recycling as scratch tapes;

- End-of-file markers, which use the date field with a special value,
- function incorrectly.

9. File or Directory Creation Test

These tests check for observed problems with file or directory creation when the file name is based on an incorrect year date. These tests apply to any system, which can store information collected from the manufacturing process, or allow for editing and creating files. Errors can occur when the system attempts to create data files after the year 2000 and the date rollover is incorrect. Systems, which create file names based on the time and date, have been observed to lockup when the file name contained non-printable or illegal characters. Another possibility exists when the file already exists and is being updated. Most user interfaces prompt for verification, "Do you wish to replace file foo.dat 12/30/99 with file foo.dat 01/03/00?" Will the newest file replace the older file in both the prompt and the actual replacement? Is the file identifier "FILE00" assumed to be older than "FILE99"?

A. File - Creation 2000

Test:

- Set the date of the system under test to a date beyond January 1, 2000.
- Create an event or choose a time such that the system will attempt to create a file.

Expected Result:

- Verify that the new file was created.
- Verify that time stamped information is valid inside any history or log files.
- Verify that any history or the application or system can use log files.

Examples:

- Systems, which store historical data that create files using a name, based on the date.

B. File - Replacement 1999-2000

Test:

- Create an old test file in 199X with data identifiable for 199X.

- Set the date of the system under test to a date beyond January 1, 2000.
- Create a new test file with the same name and new data.

Expected Result:

- Observe the prompt for the correct order of replacement, and replace old file with new file.
- Verify that the old file was replaced with the new file.
- Verify that the file contains the new data.

Examples:

- Systems, which prompt for confirmation during, file updates, such as file managers.

C. File - Replacement 2000-2000

Test:

Create an old test file in 2000 with data identifiable for 2000.
Set the date of the system under test to a date beyond the creation date of the old file.
Create a new test file with the same name and new data.

Expected Result:

- Observe the prompt for the correct order, replace old with new file
- Verify that the old file was replaced with the new file.
- Verify that the file contains the new data.

Examples:

- Systems, which prompt for confirmation during, file updates, such as file managers.

10. Audit Log Test

This test checks for problems with audit logging systems. This test applies after the year 2000 and observing the alarm display pages or reports for correct ordering of the alarm data. Failure examples include monitoring packages, which place new alarms at the end of the list after 2000. One test is to query for all items from now until 12-31-1999 and observe the results, then query for all items from now until 1-2-2000 and observe the results. Some systems fail this test by not returning any records on the second query.

A. Report - Query

Test:

- Set the date of the control system under test to a date beyond January 10, 2000.
- Create new data by forcing a fault or some system event, which will create test records.
- Set the date of the control system under test to a date beyond March 1, 2000.
- Create a new report containing the Year 2000 data by choosing four time spans:
  a) November 15, 1999 to December 31, 1999 (1999 data);
  b) November 15, 1999 to March 1, 2000 (all data);
  c) January 1, 2000 to March 1, 2000 (2000 data);
  d) February 1, 2000 to March 1, 2000 (no data).

Expected Result:

- Verify that the report with all data, b), contained all the data in the report.
- Verify that the data was ordered correctly in the report.
- Verify that the report with no data, d), executed correctly and no data was printed in the report.

B. Report - Sort

Test:

- Set the date of the control system under test to a date beyond January 1, 2000.
- Create new data by forcing a fault or some system event, which will create test records.
- Create a new report containing the Year 2000 data by choosing a valid time span.

Expected Result:

- Verify that the new data was ordered correctly in the report.

C. Report - Merge

Test:

- Set the date of the control system under test to a date beyond January 1, 2000.
- Create new data by forcing a fault or some system event, which will create test records.

- Create a new report containing the Year 2000 data by merging new data.

Expected Result:

- Verify that the new data was merged correctly in the report.

D. Report - Search

Test:

- Query or Search for an existing record created in the year 1999 with the current time in the year 1999.
- Query or Search for an existing record created in the year 1999 with the current time in the year 2000.
- Query or Search for an existing record created in the year 2000 with the current time in the year 2000.

Expected Result:

- Verify that all records are found as expected.

E. Log file purge Test

The log file purge test applies to manufacturing systems, which periodically purge old data to maintain file system space by deleting the oldest data. The problem identified occurs when after the year 2000, files with year data lower than other files (e.g. 1998 is less than 1999) are removed. Does the system consider data with a year date of 2000 to be less than 1999? If the comparison is only considering 2- digit years, this will happen and newer data can be purged.

Test:

- Verify that log data backups are available.
- Set the date of the system under test to a date beyond January 10, 2000.
- Create new data for the log file or rename some existing data.
- Attempt to purge data from the system older than 7 days.

Expected Result:

- Verify that only data from before the purge date was removed.
- VAX/VMS RMS purge command
- Database products, which support a purge function

F. Timer Test

This test verifies the creation and operation of event timers in systems or software, which provide these capabilities.

Test:

- Set the date of the control system under test prior to 2000.
- Create new timer to wake up or alarm to trigger at 10:01 AM, January 3, 2000
- Set the date of the control system under test to January 2, 2000.
- Create new timer to wake up or alarm to trigger at 10:02 AM, January 3, 2000
- Set the date of the control system under test to January 3, 2000.
- Set the time to 10:00 AM.
- Wait for the alarms to trigger

Expected Result:

- Verify that the alarm or timer created before 2000 operates correctly.
- Verify that the alarm or timer created after 2000 operates correctly.

Examples:

- UNIX chron scheduling software;
- SCADA package timer functions;
- HVAC Controls for starting and stopping ventilation or cooling equipment.

G. Input Data Test

The input data test applies to manufacturing systems, which read date information from labels or other manufacturing control systems.

Test:

- Set the date of the control system under test to January 2, 2000.
- Create input labels or simulate input from other systems with a date beyond 1-1-2000 (or 1-1- 00).
- Attempt to read the input data.

Expected Result:

- Verify that the system correctly reads the input data.

H. Output Data Test

The output data test applies to manufacturing systems, which write date information to labels or other manufacturing control systems. Systems, which print results to a printer or operator display, have been found to lock up or fail to display data after the year 2000. The cause is that from the year date rollover, invalid characters can be produced.

Test:

- Set the date of the control system under test to a date beyond January 1, 2000.
- Attempt to output data.

Expected Result:

- Verify that the data was output correctly.

Examples:

- Printed labels or product markings;
- Transfer data to other systems.

11. Activation/Deactivation Tests

This test applies to manufacturing systems, which contain passwords, accounts, or complex software license systems, which contain expiration functions.

A. Valid access

Test:

- Check that the expiration date extends past January 1, 2000
- Set the date of the system under test to a date beyond January 1, 2000.
- Attempt to execute the software licensed, or use the affected password, account, etc.

Expected Result:

- Verify that the software executes properly after January 1, 2000.

B. Expired access

Test:

- Check the expiration date.

- Set the date of the system under test to a date after the expiration date in the year 2000 or beyond.
- Attempt to execute the software licensed, or use the affected password, account, etc.

Expected Result:

- Verify that the software does not execute after the expiration date.

C. Display Data Tests

The display data test applies to manufacturing systems, which display date information on several different pages. The test must include moving the date ahead to the year 2000 and observing every screen, which the controller contains. The non-compliance can range from partial display to complete control system lockup. Many industrial controllers have unique software for each display screen, and may behave differently on any screen. Examples of this include: CNC controllers, which may have one set of display pages for tool management, another set of display pages for fault annunciation, and another for the file system, all of which may have software written by different persons or teams of persons over the product development life cycle.

Test:

- Create a list of all the date fields on all the display screens.
- Set the date of the system under test to a date beyond January 1, 2000.
- Create new files or fault records.
- Attempt to display all date fields on all display screens for file dates or fault time stamps.

Expected Result:

- Verify that each date field displays correctly.

Examples:

- A software application supports 4-digit dates in the 20xx range using a DBMS but only passes 2-digit years to the DBMS which defaults to 19xx and stores the wrong date.

12. Indirect Date Usage Tests

These tests apply to systems, which use date information in an indirect manner. The following list is intended to stimulate questions about a system, which could use the date in functions that do not require date information, but may have been implemented, using a date function.

Test:

- Identifying functions, which use the date indirectly, may be very difficult.

Expected Result:

- Where identified, verify correct operation in the year 2000.

Examples:

- Encryption/Decryption algorithms;
- Random Number generators;
- Communications protocols;
- Firmware.

APPENDIX D

Test Plan Review Checklist

1. Test Issues Checklist

The following sections provide a list of questions, which can be used to review a test plan for completeness.

1.1. General Integrity

1. System date can be set to high-risk dates:
   1999-12-31, 2000-01-01, 2000-02-29

2. Re-initialize from cold start on high-risk dates:
   1999-12-31, 2000-01-01, 2000-02-29

3. System date rolls over correctly to/from high-risk dates:
   1999-01-01, 2000-01-01, 2000-02-29, 2000-03-01

4. Does the programming language provide a function to obtain the system date on the host or through a time service?

5. Does this function return the correct system date value for high-risk dates (1999-12-31, 2000-01-01, and 2000-02-29)?

6. Does this function return the correct value for system date after the system date rolls over on high- risk dates (1999-01-01, 2000-01-01, 2000-02-29, 2000-03-01)?

7. Are there third-party products embedded in this application? Are all these products Year 2000 compliant?

8. Does the application code ignore values for explicit first 2 digits of year in the system date at any point in the program logic?

1.2 Date Integrity

9. Does the programming language support a data type for date values in the range 1900-01-01 to 2050- 12-31?

10. Does the application make a leap-year calculation? Do these calculations treat 2000 as a leap year and 1900 as a non-leap year?

11. Does the date arithmetic correctly calculate duration (differences) between dates, add dates and duration, compute date of week?

12. Does the application convert date values from one representation to another (e.g. YMD to Julian to base-and-offset internal)? Does software correctly convert between date representations according to the Gregorian calendar?

13. Does the application compare dates in any of its branching logic or calculation of Boolean values? Do all these comparisons produce correct results for all combinations of values with the expected ranges for dates?

15. Does the application include searching, sorting, merging, or indexing on internal tables, linked lists, or other data structures based on date variables? Do these operations perform correctly for all possible values for dates in the key variables? Does a key index, which includes a date field, produce correct sequence across dates in 19xx and 20xx?

16. Does the application represent dates in any variable as an offset from a base date/time? What is the maximum value for a date for this representation? What is the minimum value for a date for this representation (usually the base date)? Does the expected range of values for each variable using this date representation fall within these extremes?

16. Does the application use assigned values for the date from one variable to another? Are the first 2 digits of the value truncated during any assignment? Is the value in the target variable eventually used in a date manipulation, which requires the explicit 4-digit value for correct results?

17. Does the application use language features, which map a data address to more than one variable (such as REDEFINE in COBOL or COMMON in FORTRAN)? In all aliases for the same data space, does any variable ignore or truncate a value for explicit first 2 digits in the date value? Is the truncated value for date eventually used in a manipulation, which assumes that all values for date share the same first 2 digits?

18. Are constants for date values (including day, month, or year) used in any manipulation? Is the date constant intrinsic to the

functional requirements or a special value used in a "date" data type for convenience?

19. Does the application store and retrieve dates accurately for values in the range 1900-01-01 through 2050-12-31?

20. Does the application use sort/merge utilities to order file contents on date fields or use indexed file structures keyed on date fields? Is this order correct for all values of dates in the range 1900-01-01 through 2050-12-31?

21. Does the application rely on primary or alternate indices on a structured database for search, insert, update, or delete functions in which any key contains a date field? Will the index order be correct for all values for date in the range 1900-01-01 through 2050-12-31?

22. Are all date variables initialized to some convention for null value?

1.3 Explicit First 2 Digits of Year

23. Does the application use a language, toolkit, and/or application generator, which permits explicit first 2 digits in the date data types? If so, are values for first 2 digits in variables of these types supplied from external input or derived within the software logic?

24. Does the application use a DBMS or other layered (or horizontal) software product for data persistence to store and retrieve date variables? If so, can these products support explicit values for first 2 digits in any date variable stored and retrieved?

25. Does the application have external interfaces (I/O, APIs, external subprogram calls, IPCs, library routines, HMI) which contains a date variable with explicit first 2 digits of year? Does the software ignore, truncate, or write over the first 2 digits in a value in any such variable as it flows through the program logic to any other external interface? In any such flow, could any logic alter the value for the first 2 digits of a year in any manner inconsistent with generalized manipulations based on the Gregorian calendar?

26. Do all representations of date with explicit first 2 digits both internal to the application and in all interfaces satisfy the criteria for date compliance?

1.4 Implicit First 2 Digits of Year

27. Does the application use a language, toolkit, and/or application generator (including GUI builders) which permits date representation without explicit first 2 digits in the date data types? If so, are the first 2 digits derived for any manipulations, for passing a date value across any interface, or for permanent storage? If so, is the value for the first 2 digits correct for all possible values of date that each such variable can hold?

28. Does the application use constant values for date or portions of date (i.e., day, month, or year)? If so, for any constant, which is, a full date value or value for year, are the first 2 digits explicit in the value? Do all manipulations using each constant value, directly or indirectly (that is, carried via variables to other operations in the program logic), produce the correct results for all possible values for such date variables?

29. Does the application use any application-program interface (API), such as in-line SQL or IMS DML, which passes date variables? If so, for any date value supplied across this interface, does the receiving software provide a default or derived value of the first 2 digits of date? Are the rules for derivation on both sides of the interface consistent with each other for all possible values for a date in the respective fields?

30. Does the application support a user interface containing date fields without the explicit first 2 digits of that date? Are the first 2 digits of a date in each field unambiguous to a user for all possible values for that date in each such field?

31. Do all the date representations, both internal to the application and in all interfaces, satisfy the criteria for date compliance?

2. Year 2000 Testing Report Forms

The following sections explain the purpose and use of the Year 2000 Test Report and Year 2000 System Test Report Forms. By responding with your test results in these standardized formats, testing data can be shared throughout your organization.

2.1 Test Report Purpose

The purpose of the Year 2000 Test Report Form is to capture the results of Year 2000 testing on components shared throughout the corporation and to record the test results of unique systems. When testing combined components use the Combined Component Test Report, and for manufacturing systems use the Systems Test Report.

2.2 General Test Report Instructions

Result - Pass, Fail, Not Applicable
Effect - I (Inconvenient), S (Severe) or C (Catastrophic)
Severity - scale rating failure from 1 to 10 where:

1.- MILD - misspelled words
2.- MODERATE - misleading or redundant information
3.- ANNOYING - truncated names
4.- DISTURBING - some transactions not processed
5.- SERIOUS - lose a transaction
6.- VERY SERIOUS - incorrect transaction execution
7.- EXTREME - frequent very serious errors
8.- INTOLERABLE - database corruption
9.- CATASTROPHIC - system shutdown
10.- INFECTIOUS - shutdown spreads to other systems

3. Year 2000 Component Test Report Instructions

3.1 Header

Component Name - name of component as used in the inventory master naming guide.

Date - Date that the test was completed.

Location - Location of equipment.

Manpower requirements - The quantity of persons required for this test.

Total Time - Total net time in man-hours to do all of the applicable tests.

Contact Name - Name of the qualified person performing the test.

Vendor - Name of the manufacturer or vendor of the hardware/software component.

Model - Model name that is used in the inventory process.

Version - Name/Number of the version of the component.

3.2 Tests

Result - Enter P (Pass), F (Fail) or N/A (for Not Applicable).

Pass if the expected results are observed. Fail if abnormal results occur. N/A if the test is not applicable.

If the test fails, complete the comment field for that test to document the issues according to the instructions below.

Result Comments - If result was Pass or N/A then comments are not obligatory.

If the result was fail then the comments are obligatory to describe the failure.

Effect - Code to describe the effect of failure.

If the Result was Pass then input either D (for Date) or N (for No date) in the Effect field depending on whether a date function was found.

If the result was Fail then input either I (Inconvenient), S (Severe) or C (Catastrophic) in the Effect field.

If the result was Not Applicable, do not enter anything.

3.2.1 Year 2000 Component Test Report Form

Reference Number	Test Name	Results	Effect	Result Comments
2	Rollover, Reboot, Day of week			
2A	Rollover - 1999 to 2000 - Power on			
2B	Day of Week			
2C	Reboot – Date Retention			
2D	Rollover 1999 to 2000, Power off			
3	Manual data set			
3A	Date Set – 1 Jan 2000			
3B	Date Set – Date retention			
3C	Date Set – 29 Feb 2000			
3D	Leap Year Test			
3E	Leap Year –rollover 2/28			
3F	Leap Year – reboot 2/29			
3G	Leap Year – rollover 2/29			
4	Date Window tests			
4A	Date Window test below limit			
4B	Date Window test above limit			
4C	Date Window test change limit			
5	Other date tests			
5A	DOY – 29 Feb 2000			
5B	DOY – 31 Dec 2000			
5C	DOY – invalid dates			
6	Arithmetic date tests			
6A	Days in 2000			
6B	Days across 1999/2000			
6C	Days across leap year			
7	Upload / download tests			
7A	Upload			
7B	Download			
8	Special value test			
9	File or Directory Creation test			
9A	File Creation 2000			
9B	File – replacement 1999-2000			
9C	File – replacement 2000-2000			
10	Audit Log test			
10A	Report			
10B	Report – Sort			

10C	Report – Merge			
10D	Report – Search			
10E	Log file purge test			
10F	Timer test			
10G	Input data test			
10H	Output data test			
11	Activation/deactivation test			
11A	Valid access			
11B	Expired access			
11C	Display data tests			
11D	Indirect date usage tests			

3.2.2 Year 2000 Combined Component Test Report Purpose

The purpose of the Year 2000 Combine Component Test Report Form is to capture the results of Year 2000 testing on combined components and to record the test results.

3.2.3 Year 2000 Combined Component Test Report Instructions

Plant/Site Name - Name by which the plant or site is commonly known.

Contact Name - Name of the qualified person performing the test.

Date - Date that the test was completed.

Location - Location of equipment.
Area - Name of the particular manufacturing area within the site. Use real names not acronyms.

Combine Components Name - unique name of the combined component.

Overall Result (1-10) - severity of failure as documented

Overall Result Comments - any comments regarding overall failures.

Manpower requirements - The quantity of persons required for this test.

Total Time - Total net time in man-hours to do all of the applicable tests.

3.2.4 Subheader

Identify each component active in the test:
Vendor - Name of the manufacturer or vendor of the hardware/software component.

Model - Model name that is used in the inventory process.

Version - Name/Number of the version of the hardware/software component.

Component Type - one of the standard component types.

3.2.5 Tests

Result - Enter P (Pass), F (Fail) or N/A (for not applicable)

Pass if the expected results are observed.

Fail if abnormal results occur. N/A if the test is Not Applicable.

If the test fails, complete the comment field for that test to document the issues according to the instructions below.

Result Comments - If result was Pass or N/A then comments are not obligatory, but if the result was fail then the comments are obligatory to describe the failure.

Effect - Code to describe the effect of failure.

If the Result was Pass then input either D (for Date) or N (for No date) in the Effect field depending on whether a date function is found.

If the result was Fail then input I (Inconvenient), S (Severe) or C (Catastrophic) in the Effect field. If the result was Not Applicable, do not enter anything.

3.2.6 Year 2000 Combined Component Test Report Form

Year 2000 Combined Component Test Report Form is the same as Year 2000 Component Test Report Form, used a combined / summarized report

3.2.7 Year 2000 System Test Report Purpose

The purpose of the Year 2000 System Test Report Form is to capture the results of Year 2000 testing on manufacturing systems and to record the test results.

3.3 Year 2000 System Test Report Instructions

3.3.1 Header

System Name - Unique name of manufacturing system used in the inventory.

Date - Date that the test was completed.

Location - Location of equipment.

Manpower requirements - The quantity of persons required for this test.

Total Time - Total net time in man-hours to do all of the applicable tests.

Plant/Site Name - Name by which the plant or site is commonly known.

Area - Name of the particular manufacturing area within the site. Use real names not acronyms.

Contact Name - Name of the qualified person performing the test.

Overall Result (1-10) - severity of failure as documented

Overall Result Comments - any comments regarding overall failures.

3.3.2 Subheader

List each component and combined component active in the test.

Item - identifier to link failures to components

Vendor - Name of the manufacturer or vendor of the hardware/software component.

Model - Model name that is used in the inventory process.

Version - Name/Number of the version of the hardware/software component.

3.3.3 Failures Observed

List only tests which exhibit a failure during system testing:

Item - item number of component which failed

Section - Document section of test, which fails.

Test Name - name of test, which fails.

Result - Enter P (Pass), F (Fail) or N/A (for not applicable)

Pass if the expected results are observed.

Fail if abnormal results occur. N/A if the test is Not Applicable.

If the test fails, complete the comment field for that test to document the issues according to the instructions below.

Result Comments - If result was Pass or N/A then comments are not obligatory, but if the result was fail then the comments are obligatory to describe the failure.

Effect - Code to describe the effect of failure. If the Result was Pass then input either D (for Date) or N (for No date) in the Effect field depending on whether a date function is found.

If the result was Fail then input I (Inconvenient), S (Severe) or C (Catastrophic) in the Effect field. If the result was Not Applicable, do not enter anything.

Year 2000 System Test Report Form

Site	Contact Name	Date	Location	Area	System Name
Result	**Result Comments**		**Manpower**	**Time**	

Item	Vendor	Model	Version	Item	Vendor	Model	Version
1				26			
2				27			
3				28			
4				29			
5				30			
6				31			
7				32			
8				33			
9				34			
10				35			
11				36			
12				37			
13				38			
14				39			
15				40			
16				41			
17				42			
18				43			
19				44			
20				45			
21				46			
22				47			
23				48			
24				49			
25				50			

Item	Section	Test Name	Results	Effect	Result Comments

APPENDIX E

Department of Agriculture
Office of the Chief Information Officer
Year 2000 Program Office

CERTIFICATE OF YEAR 2000 COMPLIANCE

DEFINITION OF YEAR 2000 COMPLIANCE

Year 2000 means, with respect to Information Technology, that the Information Technology accurately processes date/time data (including, but not limited to, calculating, comparing, and sequencing) from, into, and between the twentieth and twenty-first centuries, and the years 1999 and 2000 and leap year calculations, to the extent that other information technology, used in combination with the information being acquired, properly exchanges date/time data with it.
-----FEDERAL ACQUISITION REGULATION 39.002

This certifies to the Office of the Chief Information Officer that the referenced systems has been assessed and is Year 2000 date compliant. For purposes of this certification, Year 2000 compliance includes information technology, or telecommunication, or vulnerable systems and processes (building's and facilities, or scientific and laboratory equipment.)

SYSTEM NAME: _____

AGENCY: _____

EXECUTIVE SPONSOR: _____

DATE _____

APPENDIX F

OCIO Contracts for Advisory and Assistance Services Memorandum

USDA

United States Department of Agriculture

Office of the Assistant Secretary for Administration

1400 Independence Avenue SW

Washington, DC 20250-0103

TO: Anne F. Thomson Reed
Chief Information Officer

FROM: Reba Pittman Evans
Acting Deputy Assistant Secretary
for Administration

JUN 16 1998

SUBJECT: OCIO Contracts for Advisory and Assistance Services

We have reviewed the Office of the Chief Information Officer's (OCIO) request to contract for services related to ensuring Year 2000 compliance throughout USDA. The requested services are required to support the following activities through Fiscal Year 2000:

Activity	Estimated Cost
1. Remediation of USDA agency mission-critical systems	$10 million
2. Testing of USDA agency mission-critical systems for Year 2000 compliance	$20 million
3. Year 2000 contingency planning	$ 5 million

OCIO does not plan to award these contracts on a sole source basis.

Based on the information provided, we have no comments on OCIO's request. We are returning your submission for approval, dissemination and further action as deemed appropriate.

Attachment

received JUN 16 1998

AN EQUAL OPPORTUNITY EMPLOYER